Cifras de Fluxo

Prof. Dr. Iaçanã Ianiski Weber

Confiabilidade e Segurança de Software

98G08-4

Agradecimentos especiais ao Prof. Avelino Zorzo e aos Autores Christof Paar e Jan Pelzl pelo material.

Índice

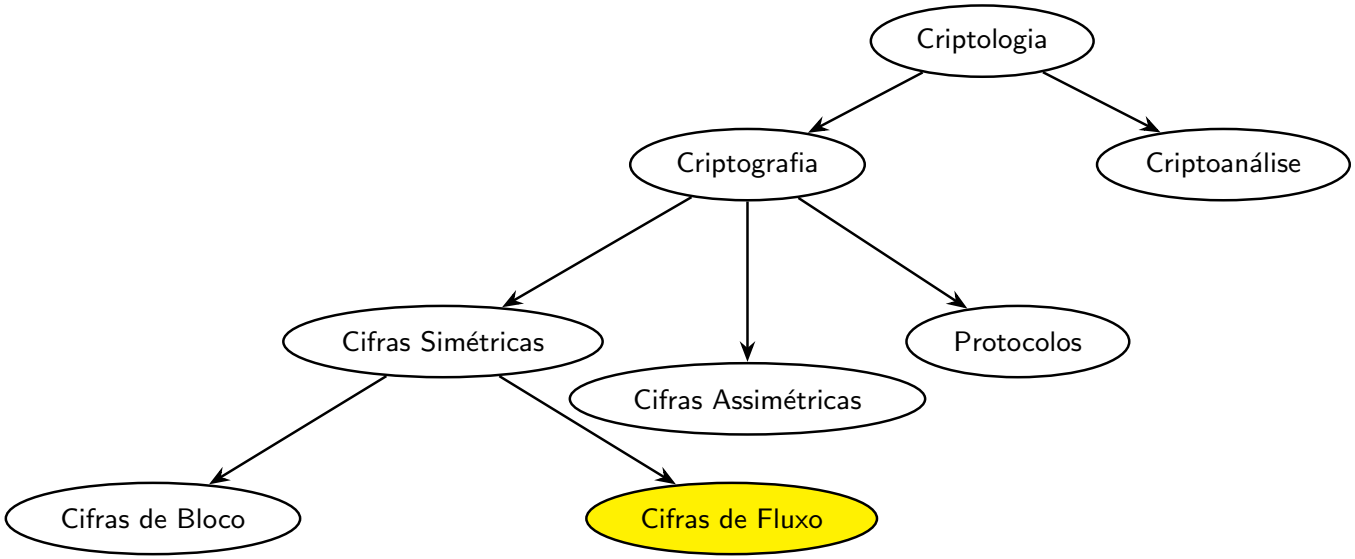
1. Introdução às Cifras de Fluxo
2. Geradores de Números Aleatórios (RNGs)
3. One Time Pad (OTP)
4. Trivium: uma cifra de fluxo moderna
5. Exercícios

Índice

1. Introdução às Cifras de Fluxo
2. Geradores de Números Aleatórios (RNGs)
3. One Time Pad (OTP)
4. Trivium: uma cifra de fluxo moderna
5. Exercícios

Cifras de Fluxo no campo da Criptologia

- Cifras de Fluxo foram inventadas em 1917 por Gilbert Vernam.



Gilbert Vernam
(1890-1960)

Cifras de Fluxo — Por que usar? (Motivação)

- **Baixa latência e streaming:** cifra bytes/bits conforme chegam; sem padding e sem conhecer o tamanho final. Ideal para tráfego interativo e telemetria.
- **Simplicidade e tempo constante em software:** implementações como *ChaCha20* evitam tabelas e *branching on secret data*, reduzindo riscos de *timing*/SCA.
- **Pegada mínima em hardware:** LFSR/NLFSR (ex. Trivium) oferecem alta vazão por área e baixo consumo — perfeito para IoT/ASIC/FPGA.
- **Modelo de erro adequado a links ruidosos:** um bit corrompido no *ciphertext* afeta apenas o bit correspondente no *plaintext* (sem efeito cascata); útil em rádio/satélite.
- **Flexibilidade de segurança:** com MAC para AE ou usa-se AEAD (p.ex., *ChaCha20-Poly1305*)
- **CSPRNG "de brinde":** o gerador de *keystream* funciona como PRG rápido e bom o suficiente para sistemas e protocolos.

Cifras de Fluxo — Onde são usadas (aplicações práticas)

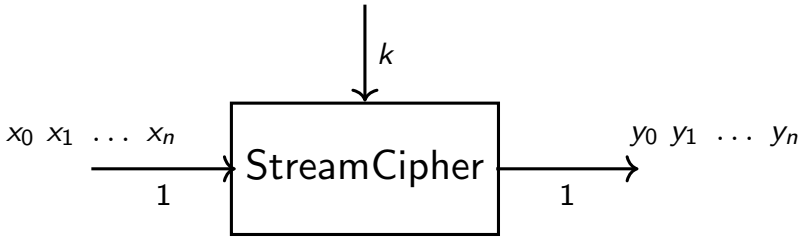
- **Protocolos de Internet e aplicativos**
  - **TLS 1.3:** *ChaCha20-Poly1305* como AEAD de primeira classe; muito usado em mobile/ARM sem AES-NI.
  - **SSH:** `chacha20-poly1305@openssh.com` amplamente implantado.
  - **VPNs:** *WireGuard* usa *ChaCha20-Poly1305* pela velocidade e simplicidade.
- **Celulares e redes sem fio**
  - **3G/4G/5G:** confidencialidade/integridade no ar com *SNOW 3G* e família *ZUC* (3GPP).
  - **Áudio/vídeo em tempo real (SRTP):** cifras em fluxo (ou modos de fluxo com blocos) para manter latência baixa.
- **Embarcados/IoT e dispositivos restritos**
  - **Baixa área:** *Trivium* em FPGA/ASIC e MCUs compactos (medidores, sensores, atuadores).
  - **Atualização de firmware:** quando código pequeno e tempo constante importam; *ChaCha20* é comum em MCUs sem AES acelerado.

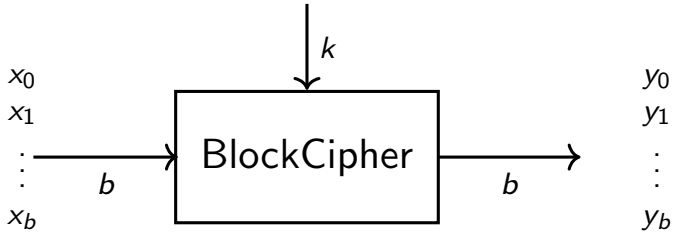
Cifras de Fluxo — Onde são usadas (aplicações práticas)

- **Satcom / enlaces de rádio**
  - **Telemetria e comando contínuos:** erros de bit ficam localizados e a sobrecarga por pacote é muito baixa.
- **Sistemas operacionais / RNGs**
  - **CSPRNGs do SO e PRNGs de usuário:** projetos baseados em *ChaCha20* pela velocidade e robustez.

- **Nota de Legado (não usar em projetos novos)**
  - **RC4:** foi onipresente (TLS/WEP), mas está **obsoleto** devido a vieses e vulnerabilidades graves.

Cifra de Fluxo vs. Cifra de Bloco

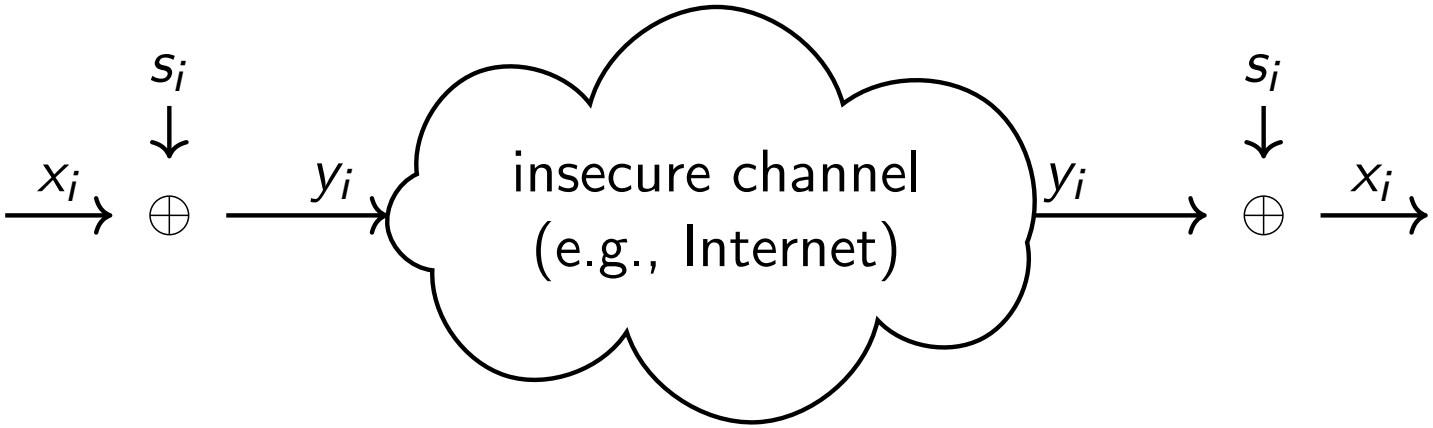




- **Cifras de Fluxo**
  - Encriptam bits individualmente
  - Geralmente pequenas e rápidas $\rightarrow$ comuns em sistemas embarcados (ex.: A5/1 em celulares GSM)
- **Cifras de Bloco**
  - Sempre encriptam blocos completos (vários bits)
  - Comuns em aplicações de Internet

Encryption and Decryption with Stream Ciphers

O Plaintext x_i, ciphertext y_i e o key stream s_i são bits individuais.



- *Encryption* e *decryption* são simples adições em módulo 2 (aka XOR).
- *Encryption* e *decryption* são a mesma função.

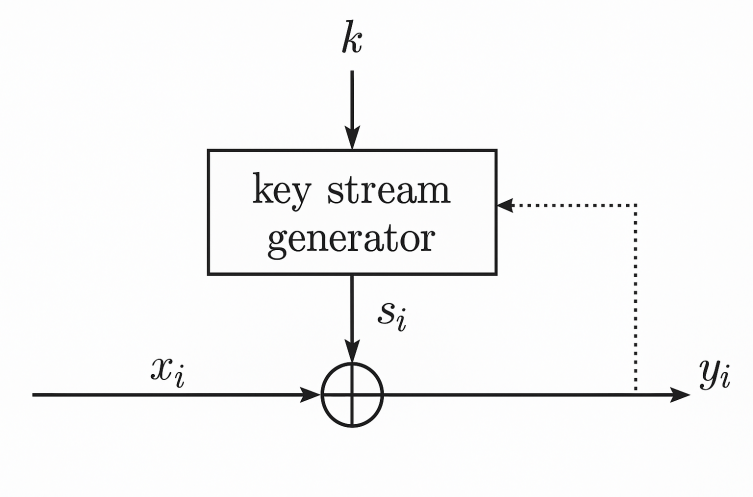
Encryption:

$$y_i = e_{s_i}(x_i) = x_i + s_i \mod 2 \qquad x_i, y_i, s_i \in \{0, 1\}$$

Decryption:

$$x_i = e_{s_i}(y_i) = y_i + s_i \mod 2 \qquad x_i, y_i, s_i \in \{0, 1\}$$

Cifra de Fluxo Síncrona vs. Assíncrona



- A segurança da cifra de fluxo depende inteiramente do fluxo de chave s_i:
  - Deve ser **aleatório**, ou seja, $\Pr(s_i = 0) = \Pr(s_i = 1) = 0.5$
  - Deve ser **reproduzível** tanto pelo transmissor quanto pelo receptor
- **Cifra de Fluxo Síncrona**
  - O fluxo de chave depende apenas da chave (e possivelmente de um vetor de inicialização - IV)
- **Cifras de Fluxo Assíncronas**
  - O fluxo de chave também depende do texto cifrado (realimentação indicada pela linha tracejada)

Por que a Adição Módulo 2 é uma Boa Função de Criptografia?

- A adição módulo 2 é equivalente à operação XOR.
- Para um fluxo de chave s_i perfeitamente aleatório, cada bit de saída do ciphertext tem 50% de chance de ser 0 ou 1.
  $\Rightarrow$ Boa propriedade estatística para o ciphertext.
- Inverter um XOR é simples, pois é a mesma operação XOR.

x_i	s_i	y_i
0	0	0
0	1	1
1	0	1
1	1	0

Cifra de Fluxo: Vazão (*Throughput*)

Comparação de desempenho de cifras simétricas (Pentium4):

Cifra	**Tamanho da chave**	**Mbit/s**
DES	56	36.95
3DES	112	13.32
AES	128	51.19
RC4 (cifra de fluxo)	(escolhível)	211.34

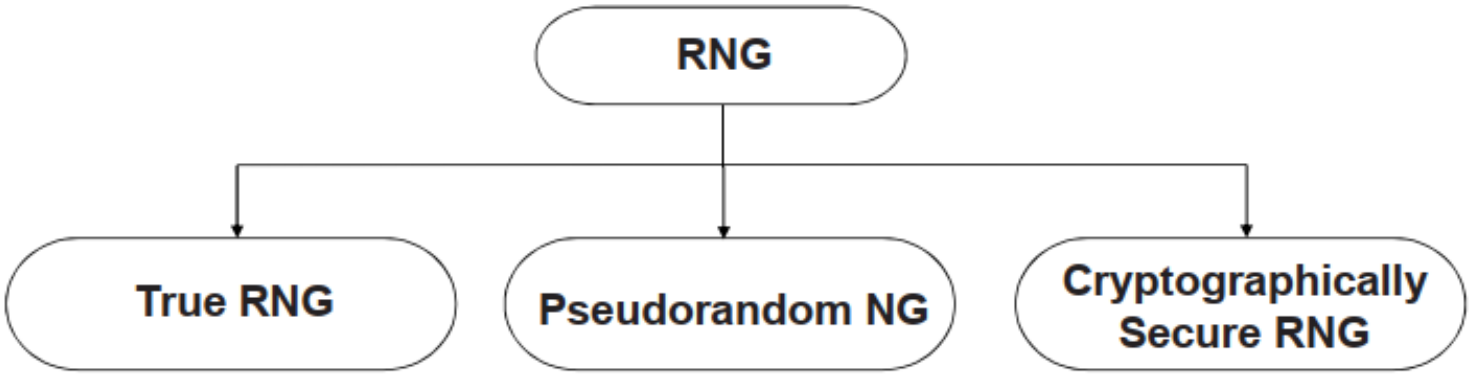
Fonte: Zhao et al., Anatomy and Performance of SSL Processing, ISPASS 2005

Índice

1. Introdução às Cifras de Fluxo
2. Geradores de Números Aleatórios (RNGs)
3. One Time Pad (OTP)
4. Trivium: uma cifra de fluxo moderna
5. Exercícios

Geradores de Números Aleatórios

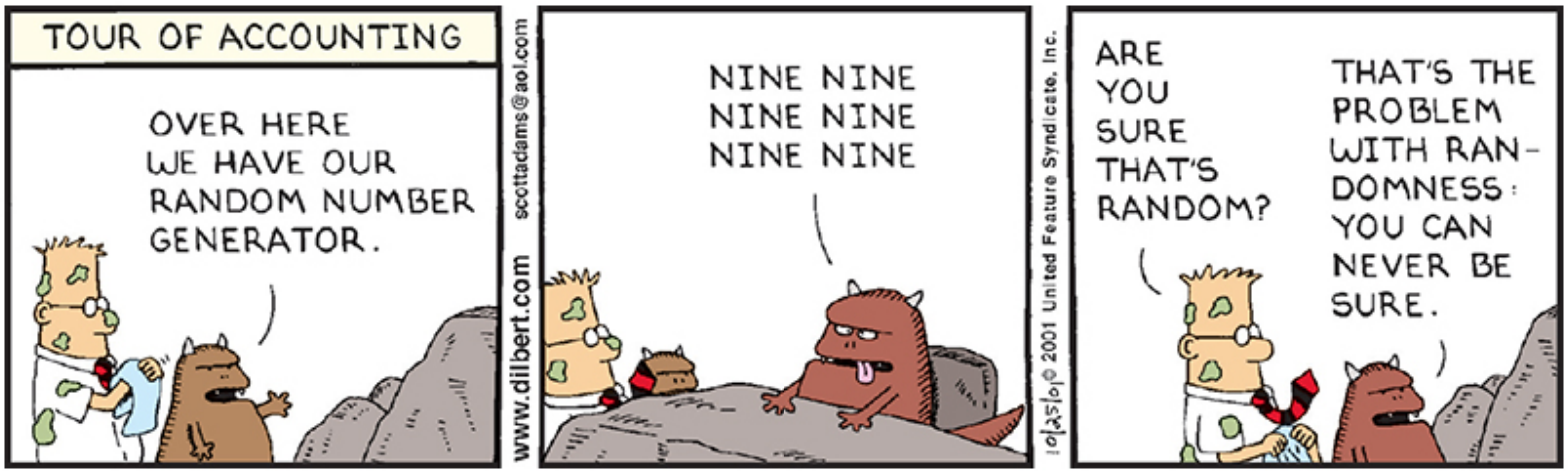
- Do inglês: random number generators - RNGs



Geradores de Números Aleatórios Verdadeiros (TRNGs)

- Baseados em processos físicos aleatórios: cara ou coroa, rolagem de dados, ruído de semicondutor, decaimento radioativo, movimento do mouse, *jitter* de clock em circuitos digitais
- A saída s_i deve ter boas propriedades estatísticas: $\Pr(s_i = 0) = \Pr(s_i = 1) = 50\%$ *(frequentemente alcançado com pós-processamento)*
- A saída não pode ser prevista nem reproduzida

Usado tipicamente para geração de chaves, *nonces* (valores de uso único) e muitos outros propósitos.



Gerador de Números Pseudoaleatórios (PRNG)

- Gera sequências a partir de um valor inicial (seed)
- Tipicamente, a saída tem boas propriedades estatísticas
- A saída pode ser reproduzida e também pode ser prevista

Muitas vezes é computado de forma recursiva:

$$s_0 = \text{seed} \qquad s_{i+1} = f(s_i, s_{i-1}, \ldots, s_{i-t})$$

Exemplo: função `rand()` em ANSI C

$$s_0 = 12345 \qquad s_{i+1} = 1103515245 \cdot s_i + 12345 \mod 2^{31}$$

A maioria dos PRNGs possui propriedades criptográficas ruins!

Criptoanalisando um PRNG Simples

PRNG Simples: **Gerador Congruente Linear**

$$S_0 = \text{seed} \qquad S_{i+1} = AS_i + B \mod m$$

Assuma:

- A, B e S_0 desconhecidos (chave secreta)
- Tamanho de A, B e S_i igual a 100 bits
- 300 bits de saída são conhecidos, por exemplo: S_1, S_2 e S_3

Solução:

$$S_2 = AS_1 + B \mod m \qquad S_3 = AS_2 + B \mod m$$

- Isso revela diretamente A e B. Todos os S_i podem ser facilmente computados!

PRNGs possuem propriedades criptográficas ruins devido à linearidade!

Gerador Pseudoaleatório Criptograficamente Seguro

- Do inglês: *Cryptographically Secure Pseudorandom Number Generator* - **CSPRNG**
- PRNG especial com a seguinte propriedade adicional:
  - A saída deve ser **imprevisível**
- **Mais precisamente:** Dado n bits consecutivos da saída s_i, os próximos bits de saída s_{n+1} não podem ser previstos (em tempo polinomial).
- Essencial para aplicações em criptografia, especialmente em cifras de fluxo.
- **Observação:** Quase nenhuma outra aplicação exige imprevisibilidade, mas muitos (muitos!) outros sistemas computacionais dependem de PRNGs.

Índice

1. Introdução às Cifras de Fluxo
2. Geradores de Números Aleatórios (RNGs)
3. One Time Pad (OTP)
4. Trivium: uma cifra de fluxo moderna
5. Exercícios

One-Time Pad (OTP)

Criptossistema de segurança perfeita:

- Um criptossistema é considerado de segurança perfeita se ele não puder ser quebrado mesmo com *recursos computacionais infinitos*.

One-Time Pad

- Um criptossistema desenvolvido por Mauborgne baseado na cifra de fluxo de Vernam.
- **Propriedades:**
  - Suponha que o texto claro, o texto cifrado e a chave consistam em bits individuais:

$$x_i,\ y_i,\ k_i \in \{0,1\}$$

Funções de Criptografia

$$\begin{aligned} \text{Encriptação:} \quad & e_{k_i}(x_i) = x_i \oplus k_i \\ \text{Decriptação:} \quad & d_{k_i}(y_i) = y_i \oplus k_i \end{aligned}$$

O OTP tem segurança perfeita se, e somente se, a chave k_i for usada apenas uma vez!

One-Time Pad (OTP)

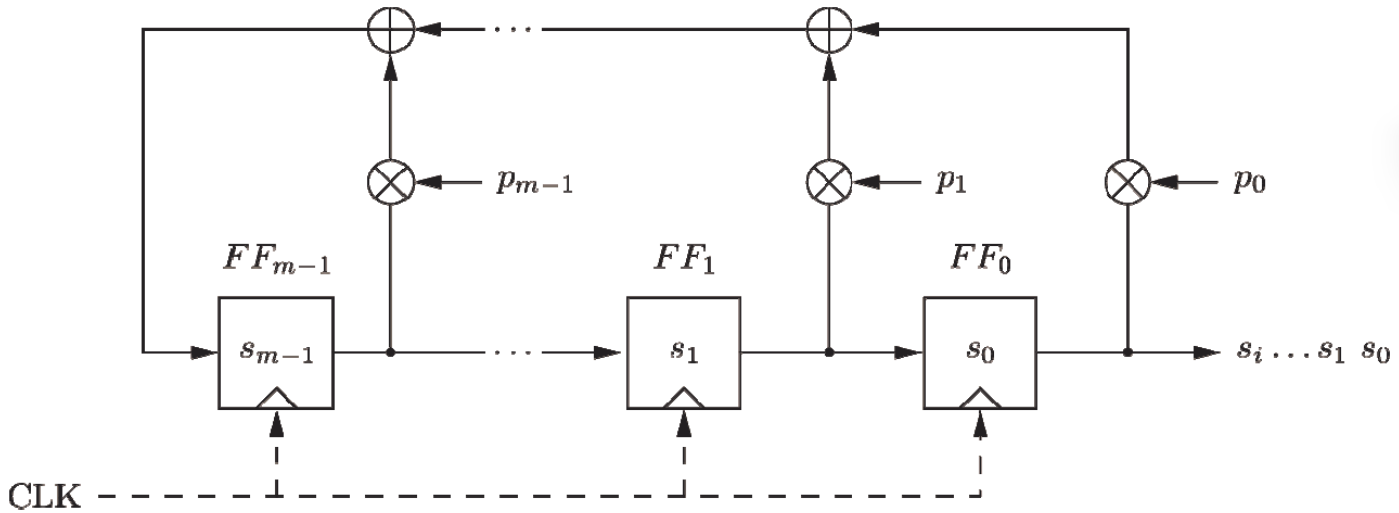
Criptossistema incondicionalmente seguro:

$$\begin{aligned} y_0 &= x_0 \oplus k_0 \\ y_1 &= x_1 \oplus k_1 \\ &\vdots \end{aligned}$$

- Cada equação é uma equação linear com duas incógnitas.
- $\Rightarrow$ Para cada y_i, os valores $x_i = 0$ e $x_i = 1$ são equiprováveis!
- $\Rightarrow$ Isso é verdade se, e somente se, $k_0, k_1, \ldots$ forem **independentes**, ou seja, todos os k_i devem ser gerados verdadeiramente aleatórios.
- $\Rightarrow$ Pode-se demonstrar que tais sistemas não podem ser resolvidos.

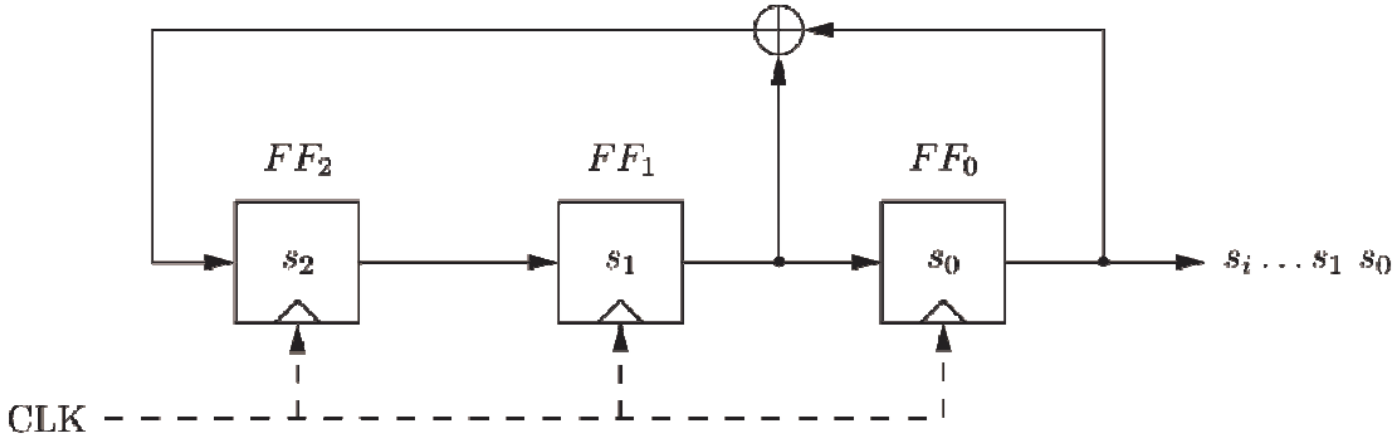
Desvantagem: Para quase todas as aplicações, o OTP é **impraticável**, pois a chave precisa ser tão longa quanto a mensagem!
(Imagine que você precise encriptar um anexo de e-mail de 1GB.)

Registradores de Deslocamento com Realimentação Linear



- Do inglês: *Linear Feedback Shift Registers* - LFSRs
- Flip-flops (FF) concatenados, formando um registrador de deslocamento com caminho de realimentação
- A realimentação calcula uma nova entrada usando a operação XOR entre certos bits do estado
- O grau m é determinado pelo número de elementos de armazenamento
- Se $p_i = 1$, a conexão de realimentação está presente ("chave fechada"); se $p_i = 0$, não há realimentação desse flip-flop ("chave aberta")
- A sequência de saída se repete periodicamente
- Comprimento **máximo** da sequência de saída: $2^m - 1$

Saída de um LFSR e Comprimento Máximo de Sequência



- A saída do LFSR é descrita por uma equação recursiva:

$$s_{i+3} = s_{i+1} + s_i \mod 2$$

- O comprimento máximo da sequência (de $2^3 - 1 = 7$) só é alcançado para certas configurações de realimentação, como a mostrada aqui.

clk	FF_2	FF_1	$FF_0 = s_i$
0	1	0	0
1	0	1	0
2	1	0	1
3	1	1	0
4	1	1	1
5	0	1	1
6	1	0	1
7	1	1	0
8	1	1	1

Segurança de LFSRs

LFSRs normalmente são descritos por polinômios:

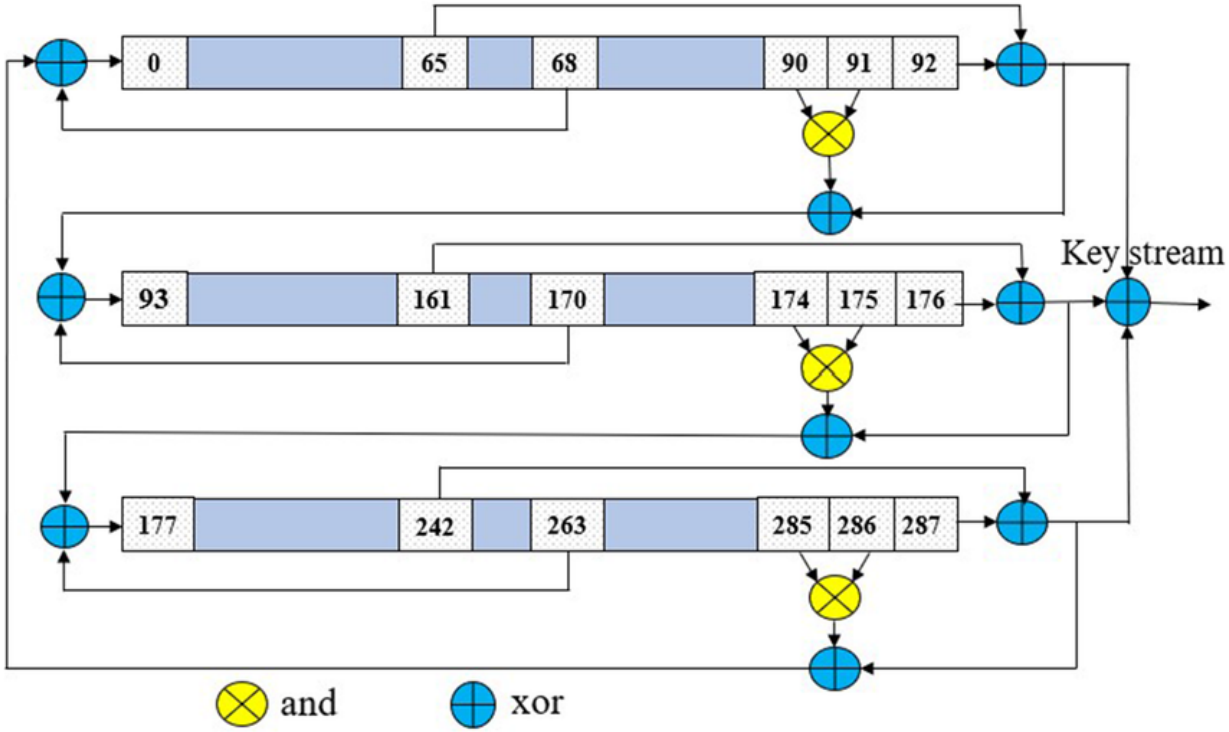
$$P(x) = x^m + p_{m-1}x^{m-1} + \cdots + p_1 x + p_0$$

- Um único LFSR gera uma saída altamente previsível.
- Se $2m$ bits de saída de um LFSR de grau m forem conhecidos, os coeficientes de realimentação p_i do LFSR podem ser encontrados resolvendo um sistema de equações lineares.
- Por esse motivo, muitas cifras de fluxo utilizam **combinações** de LFSRs.

Índice

1. Introdução às Cifras de Fluxo
2. Geradores de Números Aleatórios (RNGs)
3. One Time Pad (OTP)
4. Trivium: uma cifra de fluxo moderna
5. Exercícios

Uma Cifra de Fluxo Moderna – Trivium



- Três LFSRs *não-lineares* (NLFSR) de comprimento 93, 84 e 111.
- A soma-XOR das saídas dos três NLFSRs gera o *key stream* s_i.
- Pequeno em hardware:
  - Total de registradores: 288
  - Não-linearidade: 3 portas AND
  - 10 portas XOR (sendo 1 com três entradas)

Trivium

Inicialização:

- Carregar IV de 80 bits no registrador A
- Carregar chave de 80 bits no registrador B
- Definir c_{109}, c_{110}, $c_{111} = 1$, todos os outros bits em 0

Pré-aquecimento:

- Ciclar a cifra $4 \times 288 = 1152$ vezes sem gerar saída

Criptografia:

- A saída s_i é a soma-XOR das saídas dos três NLFSRs

O projeto pode ser paralelizado para produzir até 64 bits por ciclo de clock.

Registrador	Tamanho	Bit de Feedback	Bit de Feedforward	Entradas AND
A	93	69	66	91, 92
B	84	78	69	82, 83
C	111	87	66	109, 110

Lições Aprendidas

- Cifras de fluxo são menos populares do que cifras de bloco na maioria dos domínios, como a segurança na Internet. Existem exceções em alguns contextos, como a cifra de fluxo RC4.
- Cifras de fluxo às vezes requerem menos recursos (por exemplo, área de chip ou tamanho de código) para implementação do que cifras de bloco, sendo atrativas para ambientes restritos como embarcados.
- Os requisitos para um *gerador de números pseudorrandômicos criptograficamente seguro* são muito mais exigentes do que os de PRNGs usados em outras aplicações como testes ou simulações.
- O *One-Time Pad* é uma cifra simétrica com segurança perfeita. No entanto, é altamente impraticável para a maioria das aplicações, pois a chave precisa ter o mesmo tamanho da mensagem.
- LFSRs únicos são fracos como cifras de fluxo, apesar de suas boas propriedades estatísticas. Porém, combinações cuidadosas de vários LFSRs podem gerar cifras robustas.

Índice

1. Introdução às Cifras de Fluxo
2. Geradores de Números Aleatórios (RNGs)
3. One Time Pad (OTP)
4. Trivium: uma cifra de fluxo moderna
5. Exercícios

Exercício: Vulnerabilidade de OTP com Reuso de Chave

Considere uma cifra semelhante ao *One-Time Pad* (OTP), mas com uma chave curta de 128 bits. A chave de 128 bits é reutilizada periodicamente para cifrar grandes volumes de dados.

Tarefa:

- Descreva como um ataque pode quebrar esse esquema.

Dicas:

- Analise o que acontece quando a mesma chave é usada para cifrar duas mensagens distintas.
- Imagine que um atacante consegue o *ciphertext* e também sabe o contexto do dado, por exemplo, protocolos padronizados (e.g. cabeçalhos IP, HTTP, JSON) ou arquivos com estruturas fixas (PDF, ZIP, etc.) que têm alta redundância.

Solução: Reutilização de Chave em OTP

Suponha duas mensagens distintas m_1 e m_2 cifradas com a mesma chave k:

$$c_1 = m_1 \oplus k \qquad c_2 = m_2 \oplus k$$

Um atacante que observa ambos os ciphertexts pode fazer:

$$c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$$

Ou seja: a operação cancela a chave, e o atacante obtém o XOR das mensagens originais.

Consequência:

- O XOR entre mensagens pode revelar padrões, estruturas repetidas ou até permitir adivinhar partes do conteúdo.
- Isso é particularmente eficaz quando as mensagens têm formato previsível (ex: cabeçalhos, arquivos estruturados, padrões de linguagem natural).

Conclusão: O *One-Time Pad* só é seguro se a chave nunca for reutilizada.

Exercício: O Paradoxo da Busca Exaustiva no OTP

À primeira vista, parece que uma busca exaustiva de chave é possível contra um sistema OTP.

Cenário:

- Mensagem curta de 5 caracteres ASCII ($5 \times 8 = 40$ bits)
- Chave OTP de 40 bits
- Possibilidade de testar todas as 2^{40} chaves

Tarefa: Explique por que uma busca exaustiva **não** é bem-sucedida, mesmo que o atacante tenha capacidade computacional suficiente.

Dica: Resolva o paradoxo: o OTP é *incondicionalmente seguro*, mas o espaço de chaves é finito e pequeno. Então por que não conseguimos quebrá-lo com força bruta?

Respostas do tipo "o OTP é seguro e por isso a força bruta não funciona" não são aceitáveis.

Exemplo para Ilustrar o Paradoxo do OTP

Mensagem cifrada com OTP de 40 bits:

Exemplo de palavras com 5 caracteres (40 bits):

- `PLANO` $\Rightarrow$ `01010000 01001100 01000001 01001110 01001111`

- `FUGIR` $\Rightarrow$ `01000110 01010101 01000111 01001001 01010010`
- `PAZES` $\Rightarrow$ `01010000 01000001 01011010 01000101 01010011`

Ideia: Para qualquer `ciphertext` de 40 bits, existem várias chaves possíveis que produzem mensagens diferentes, todas com aparência plausível.

Isso mostra que uma busca exaustiva revela todas as possíveis mensagens — mas nenhuma evidência de qual é a correta.